Bakar Kiasan: Flexible Contract Checking for Critical Systems using Symbolic Execution

SAnToS Laboratory, Kansas State University, USA

http://www.cis.ksu.edu/santos

John Hatcliff
Jason Belt
Patrice Chalin
William Deng (Google Inc.)
David Hardin (Rockwell Collins Inc.)
Robby

Funding:

AFOSR
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Rockwell Collins

Context

Hi-Lite

Unit Test / Unit Proof

SPARK

Our work is targeted for SPARK and can be understood as a providing a complementary technology that can be integrated into Hi-Lite's Unit Test / Unit Proof technology.

What is SPARK?

Language and verification framework designed for critical systems

Interface Specification Language

Annotations for pre/post-conditions, assertions, loop invariants, information flow specifications

+

Programming Language

Subset of Ada appropriate for critical systems -- no heap data, pointers, exceptions, recursion, gotos, aliasing

SPARK Contracts

SPARK includes annotations for assertions, pre/post-conditions that can be used to express *software contracts*

Simple pre-condition with existential quantification…

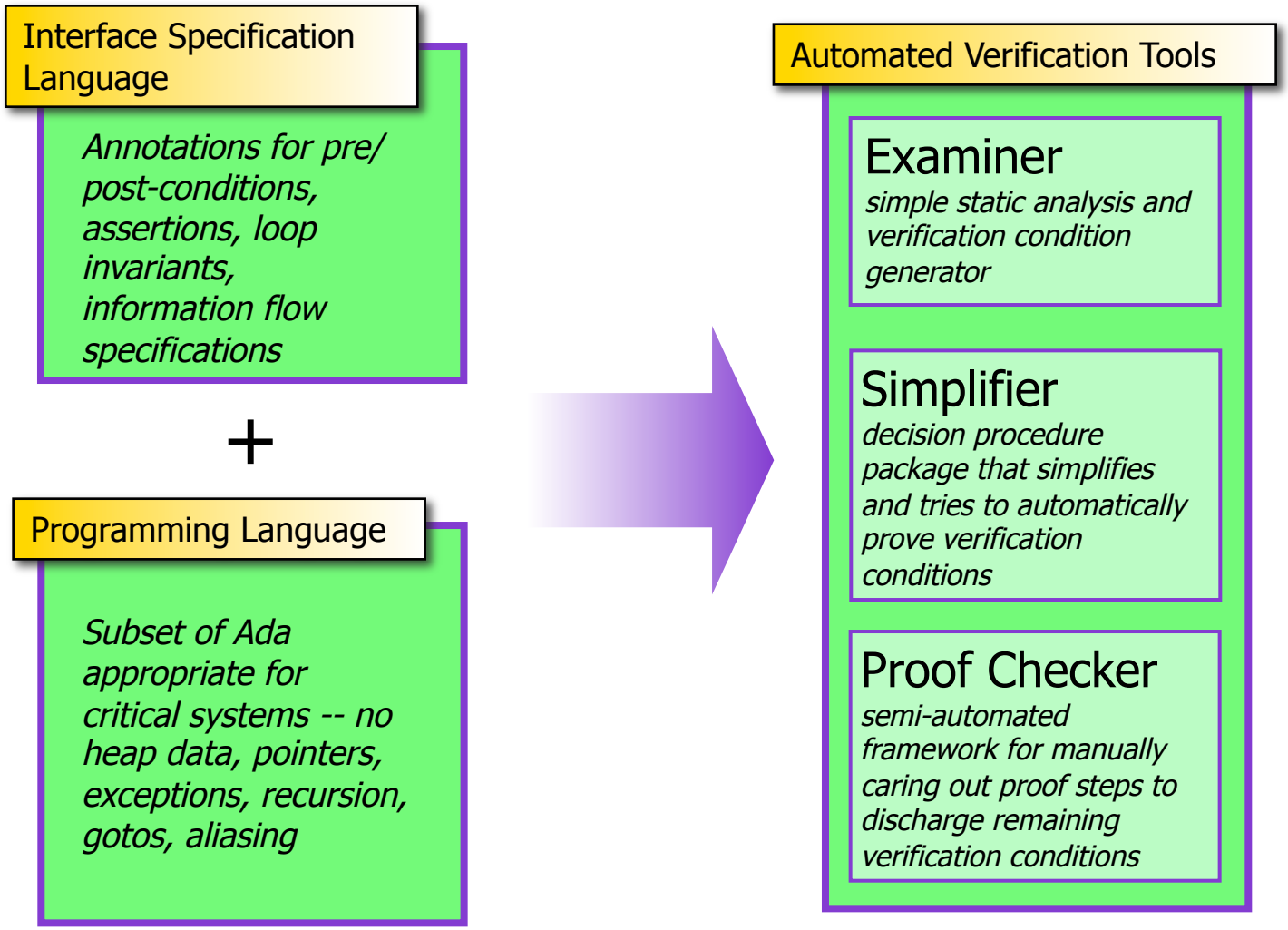
```
function FindSought
 (A: Table; Sought: Integer) return Index;
--# pre for some M in Index => ( A(M) = Sought );
--# return Z => (( A(Z) = Sought) and
--#     (for all M in Index range 1 .. (Z - 1) =>
--#          (A(M) /= Sought)));
```

Post-condition constraining return value to inputs using universal quantification…

"SPARK Examiner with Run-time Checker…", p. 22

What is SPARK?

Language and verification framework designed for critical systems



Uses of SPARK

SPARK has been (is being) used in a number of safety and security critical applications

- Tokeneer -- biometrics and smart authentication in card technology. Demonstration project sponsored by Praxis and NSA
  - http://www.adacore.com/home/products/sparkpro/tokeneer/
- Several large scale security critical projects at Rockwell Collins such as the Janus crypto-graphic engine.
- Avionics systems in the Lockheed C130J and EuroFighter Typhoon projects
- iFACTS - United Kingdom next generation air-traffic control system (team of 100+ developers at Praxis).
- [Rockwell Collins] development of certified embedded security devices

…this talk will emphasize experiences with Rockwell Collins on a DoD-funded research project that involved developing a prototype of an embedded security device

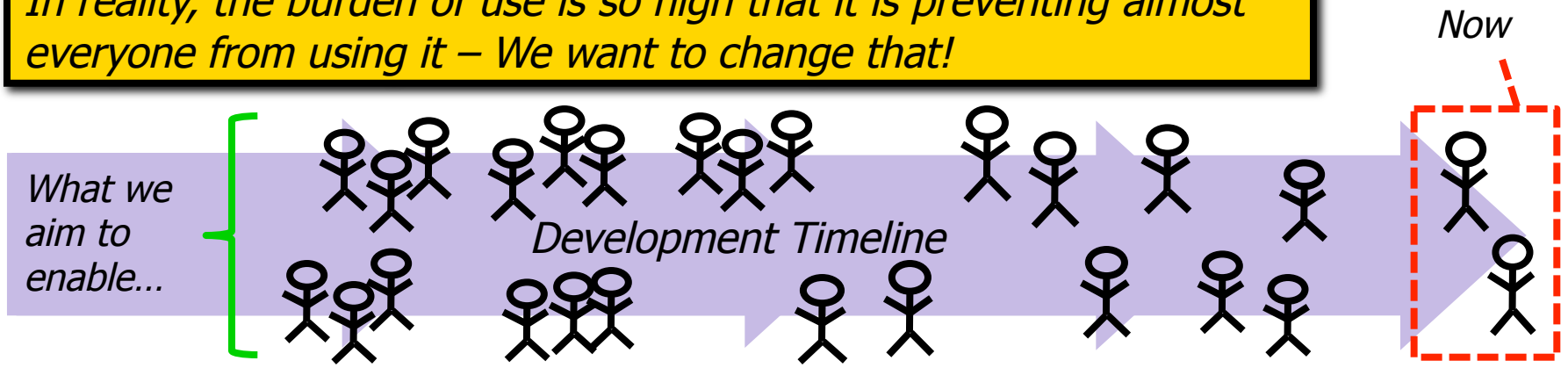
What are the obstacles?

Unfortunately, none of projects makes extensive use of SPARK's contract language beyond "type safety"

Obstacles

- Loop invariants required
- In many cases, developers get only segmented evidence of a contract's correctness (all or nothing)
- Basic behavioral properties have to be specified in a separate "proof language" (FDL)
- Technique is not connected with other quality assurance techniques (e.g., testing)

In reality, the burden of use is so high that it is preventing almost everyone from using it – We want to change that!



Our Work

Better integration of contract checking into SPARK developer workflows

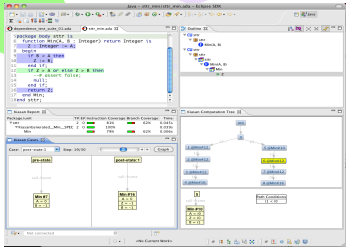
Functional Contracts (pre/post, assertions)

```
procedure Add(E: Element_Type)
 --# pre Member_Count < Size_Range'Last;
 --# post (isMember(E, Elem_Array, Member_Count) -> (
 --#       Elem_Array = Elem_Array~ and Member_Count = Member_Count~))
 --#  and (not isMember(E, Elem_Array, Member_Count) -> (
 --#       Elem_Array = Elem_Array~[Member_Count => E]
 --#       and Member_Count = Member_Count~ + 1));
```

Automatic Checking

Developers

Kiasan
Symbolic Execution Engine



SPARK Eclipse IDE

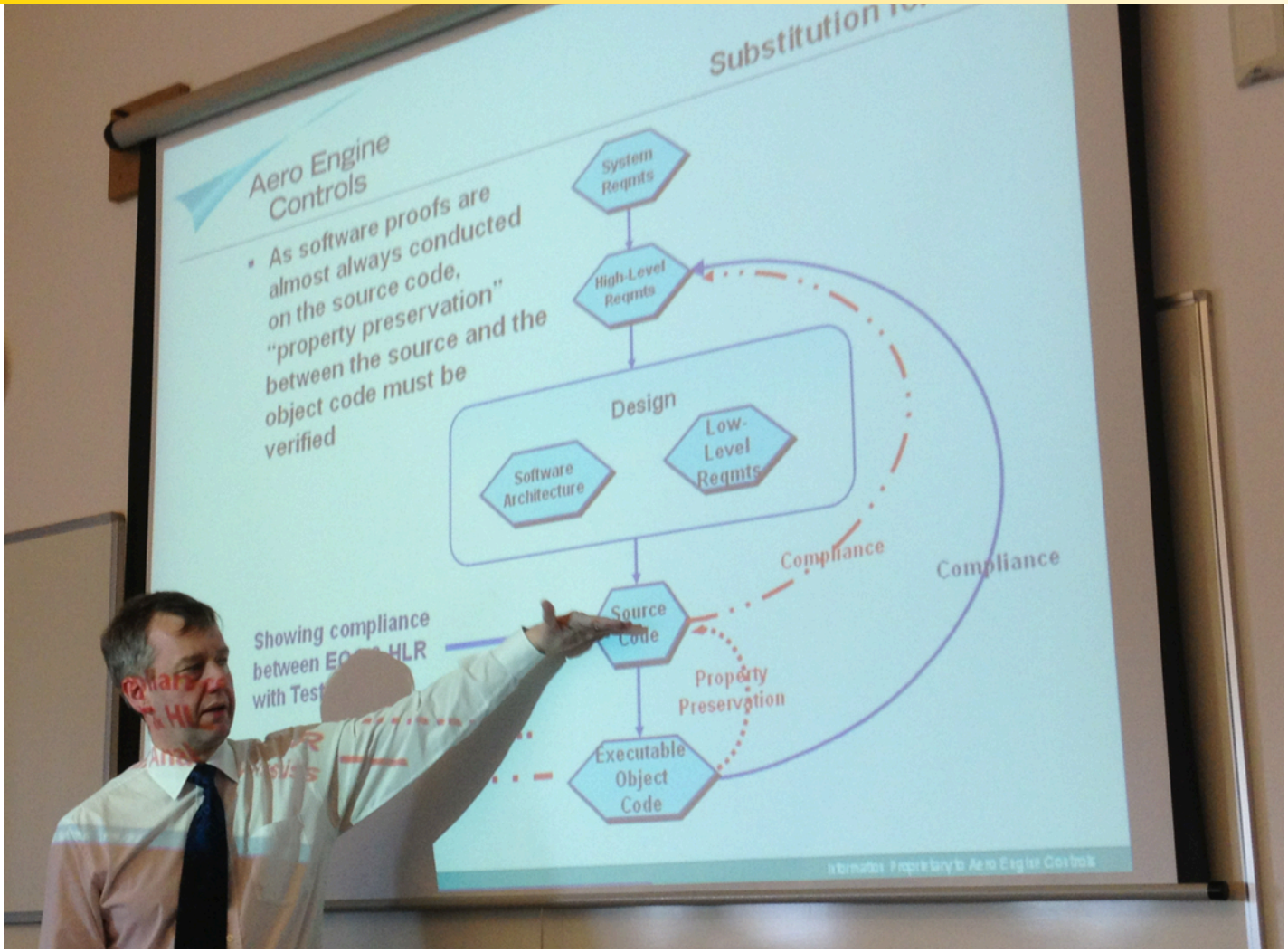
Use symbolic execution, not just for bug-finding or test-case generation, but for contract checking that complements the existing facilities of SPARK

Themes

- Highly automated; payback is on par with investment;
- Meaningful checking without loop invariants (bounded loop unfolding)
- When verification engine processes code, communicate "knowledge" gained to developer
- Keep focus on the source code level, instead of using a separate formalism like FDL
- SPARK supports only declarative contracts; we want to support both declarative and executable specs
- Connect to other quality assurance techniques; phrase results in terms of what developers already understand

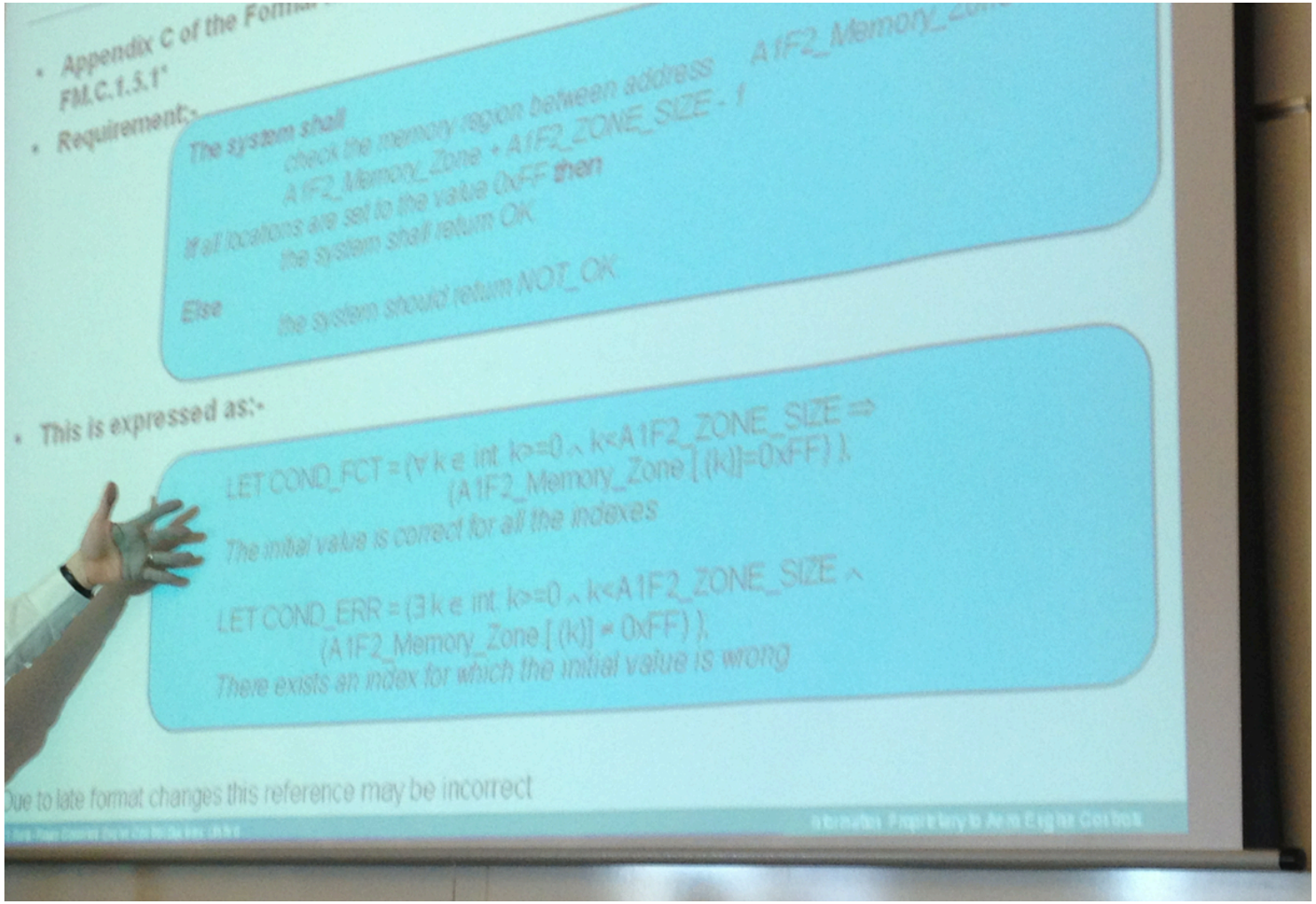
Certification Context

Duncan Brown explaining high-level requirements / low-level requirements



Certification Context

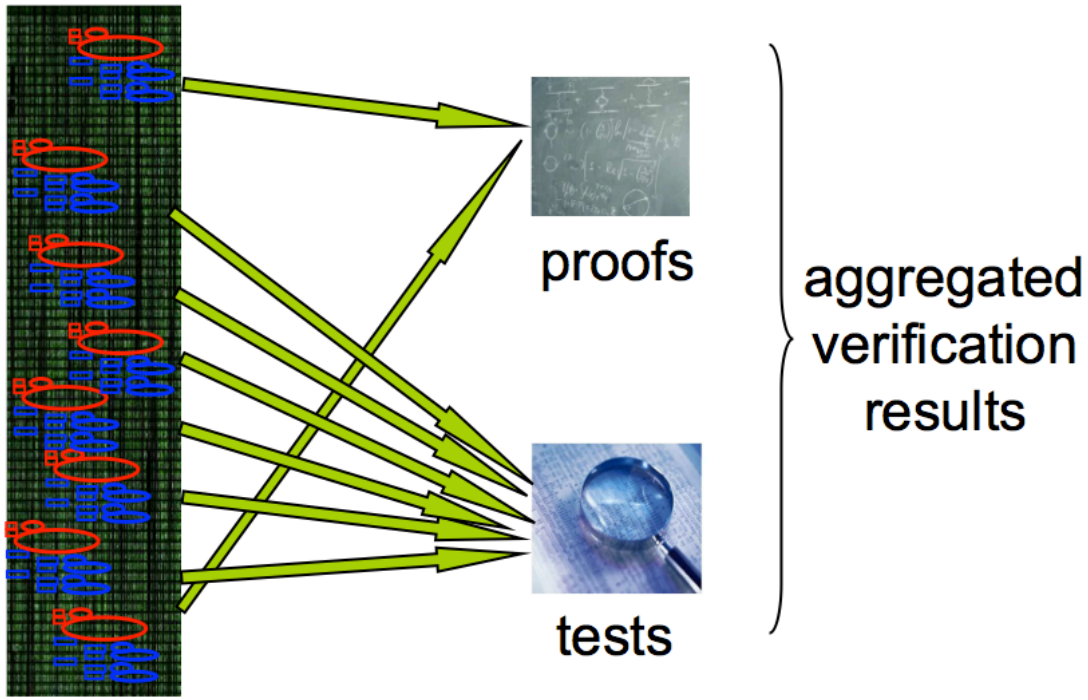
AirBus approach to replacing Unit Test with Unit Proof



Unit Test / Unit Proof

Yannick Moy

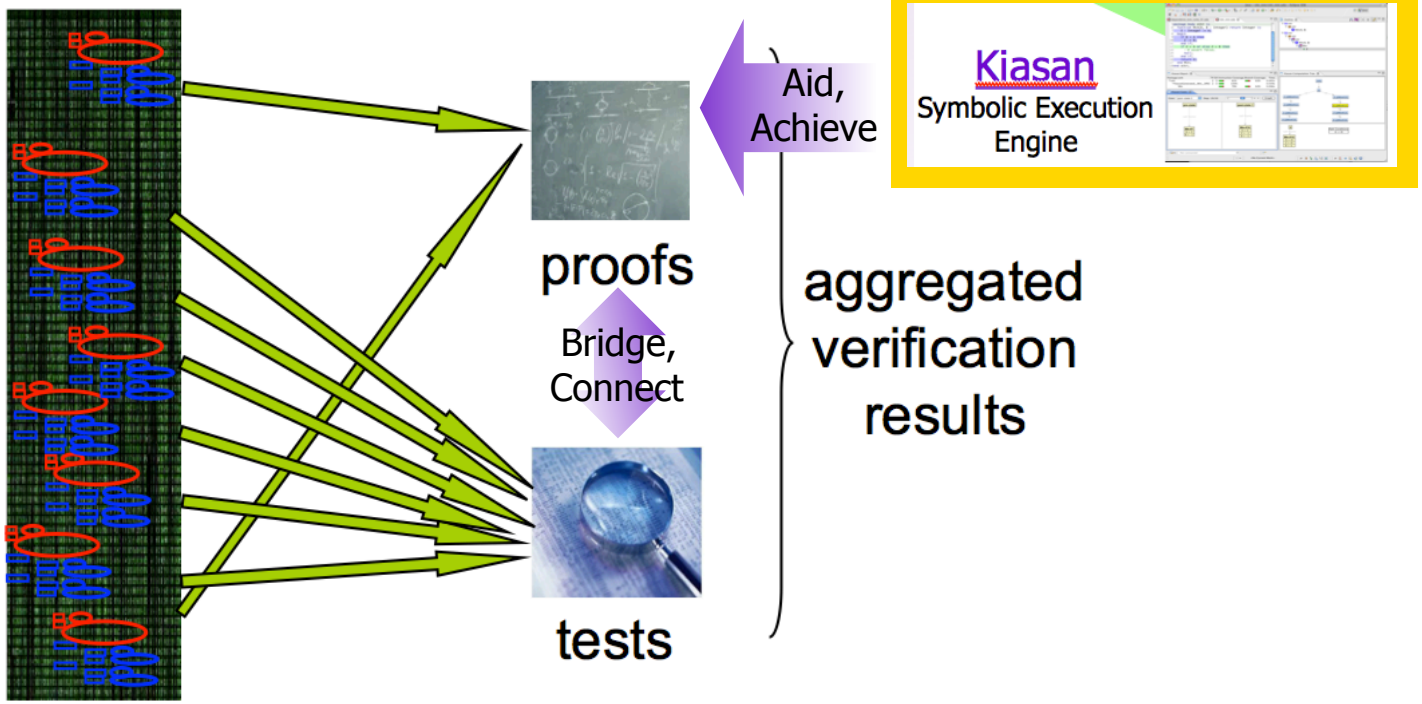
Unit Proof and Unit Test in Hi-Lite



Unit Test / Unit Proof

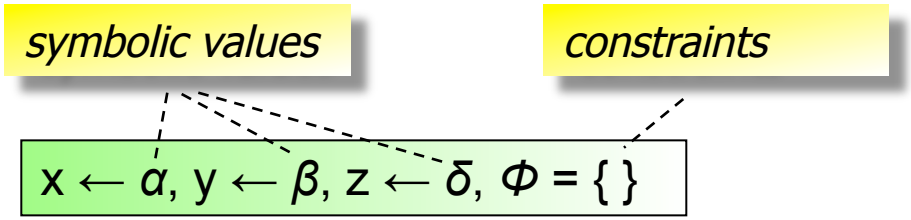
...umm, ok,
the correct Yannick

Unit Proof and Unit Test in Hi-Lite



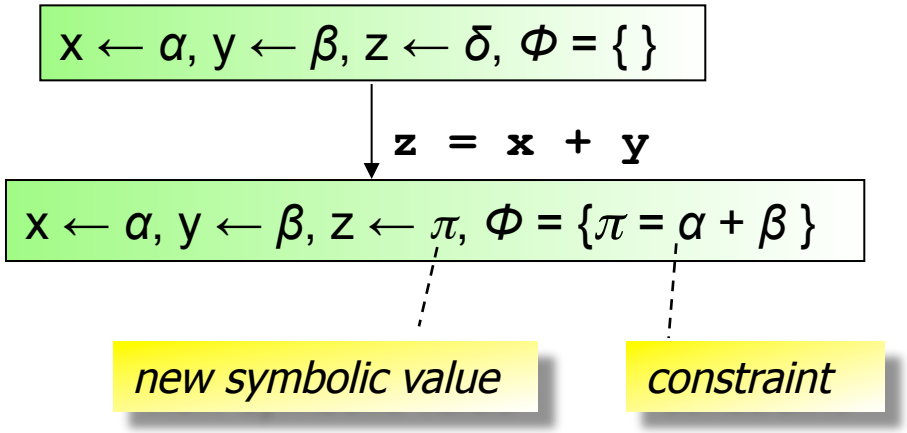
Symbolic Execution [King:ACM76]

```
void foo(int x,
         int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



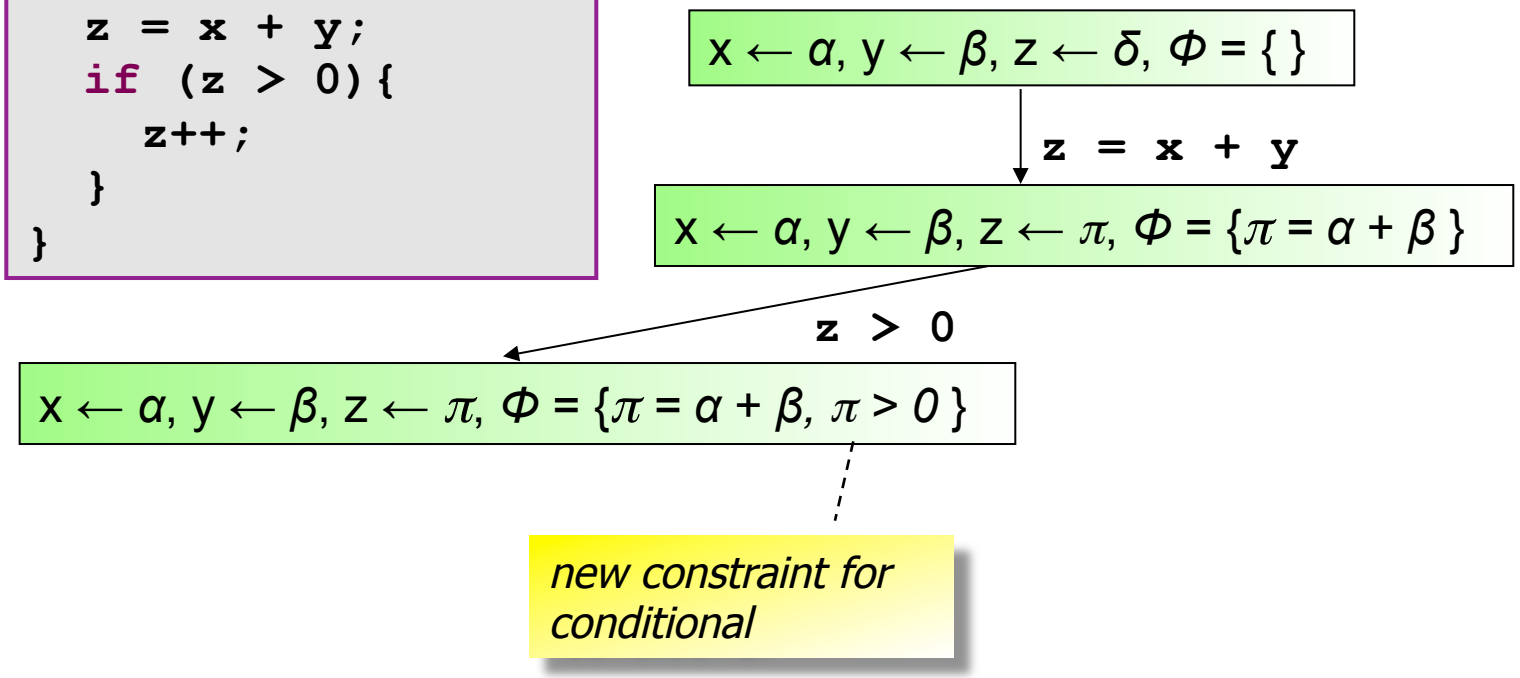
Symbolic Execution [King:ACM76]

```
void foo(int x,
         int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



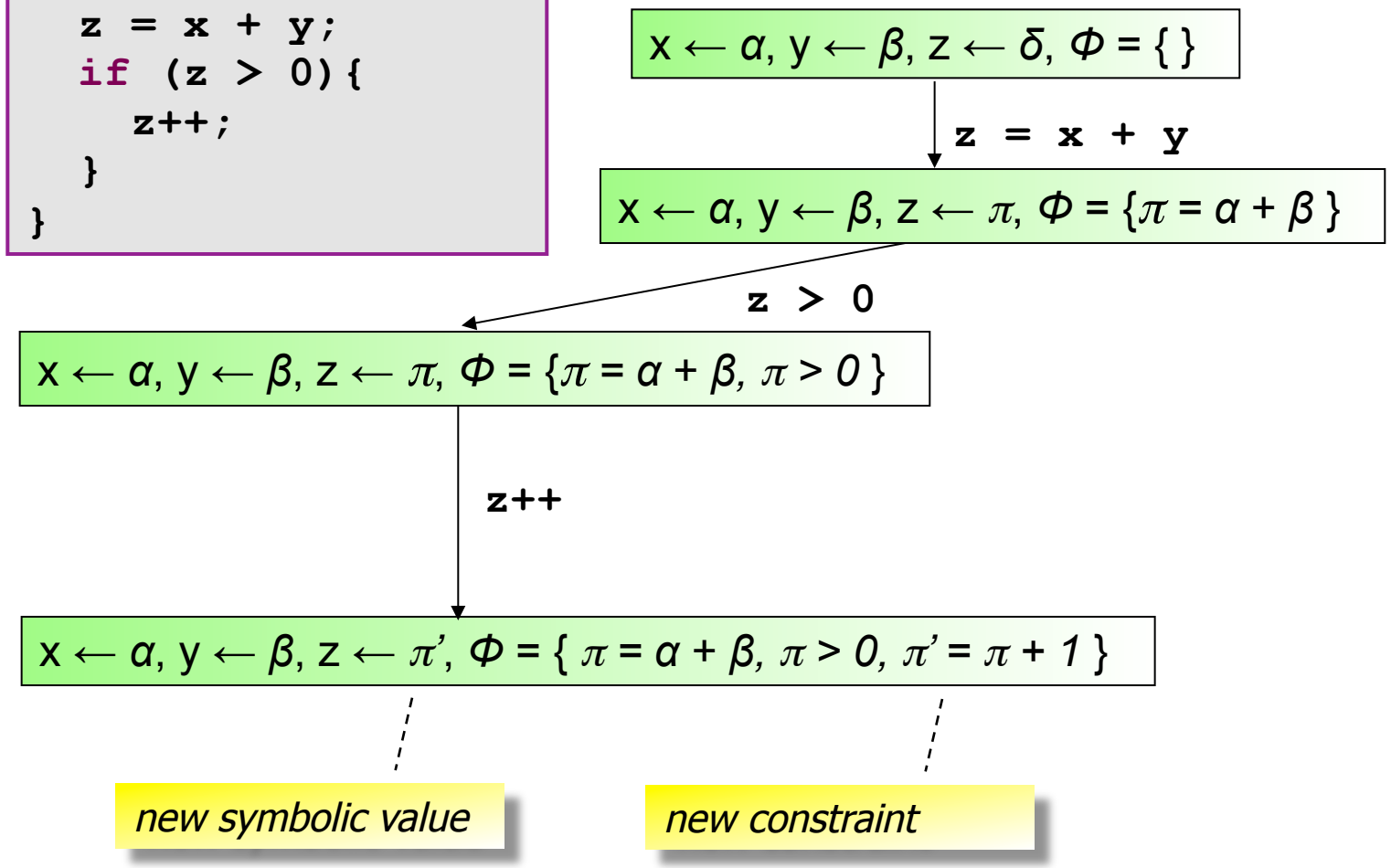
Symbolic Execution [King:ACM76]

```
void foo(int x,
         int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



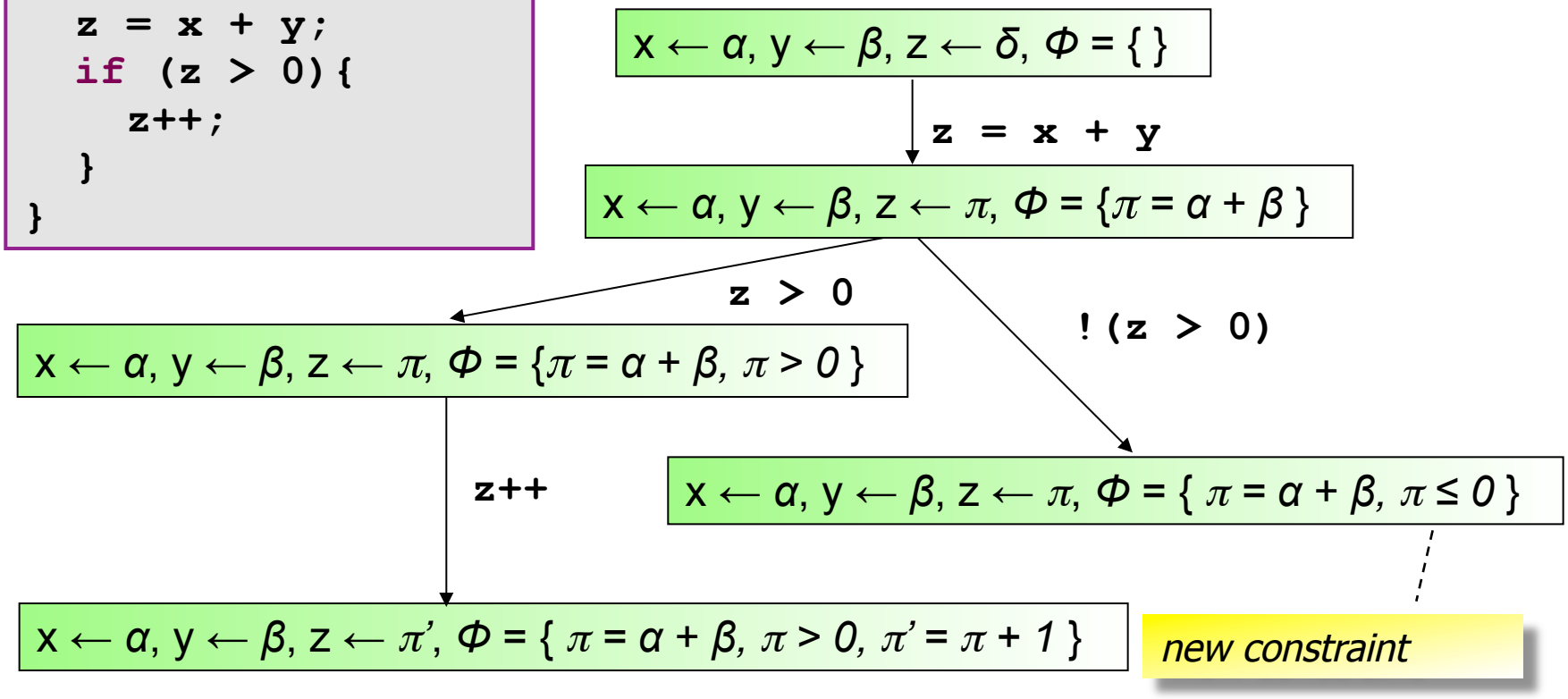
Symbolic Execution [King:ACM76]

```
void foo(int x,
         int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



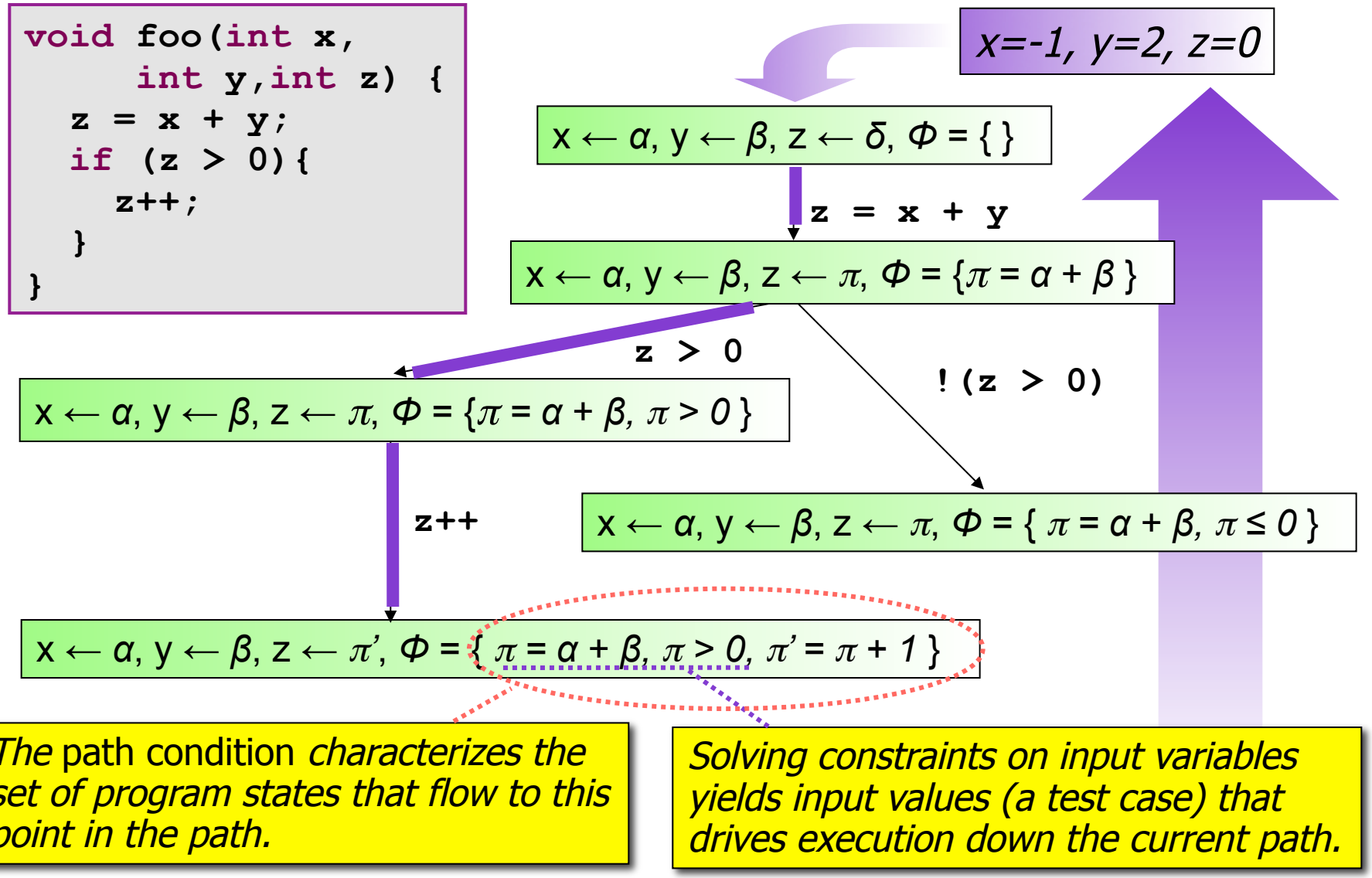
Symbolic Execution [King:ACM76]

```
void foo(int x,
         int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```

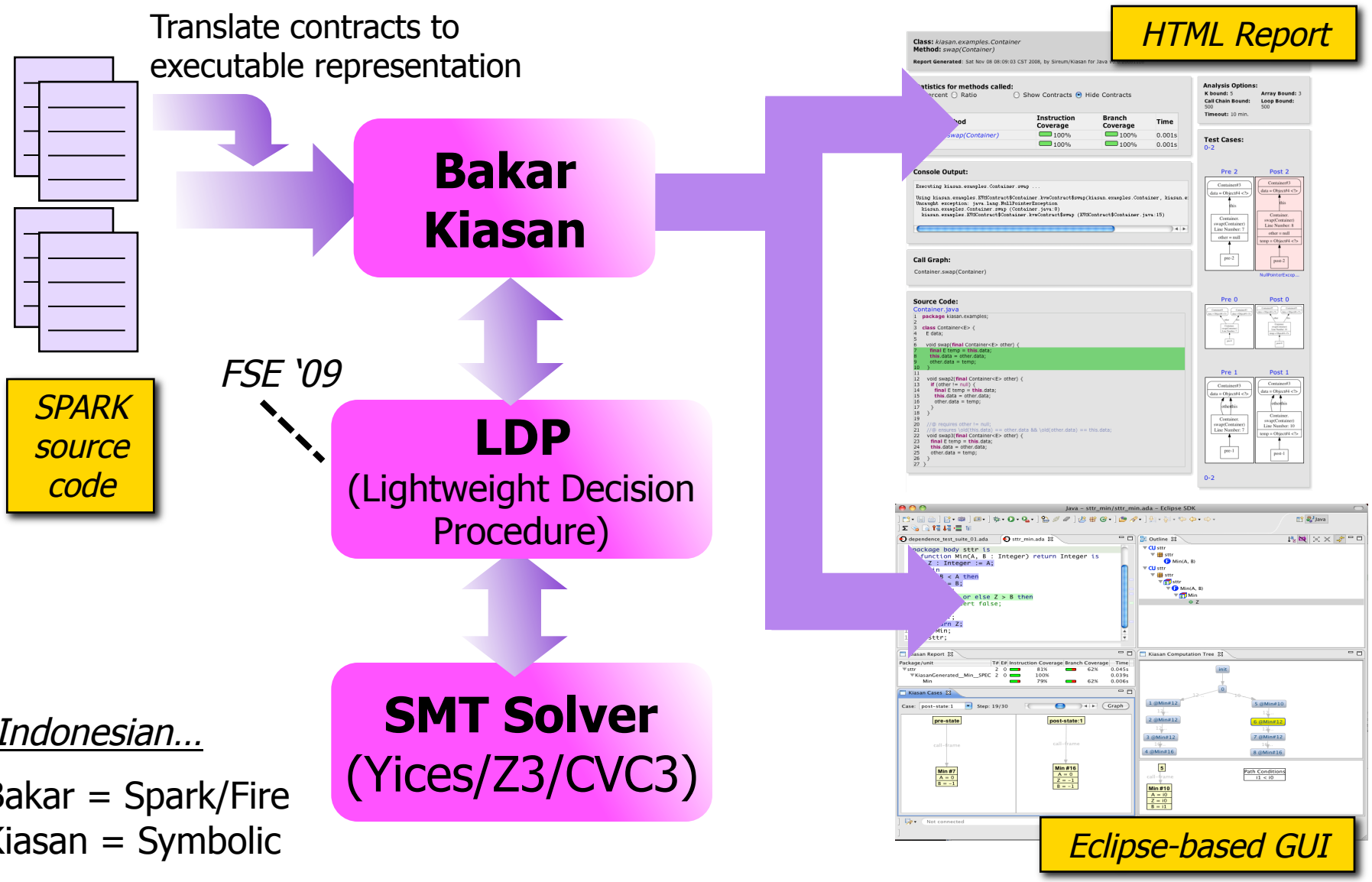


…symbolic execution characterizes (theoretically) infinite number of real executions!

Kiasan -- Solving Constraints



Bakar Kiasan Architecture



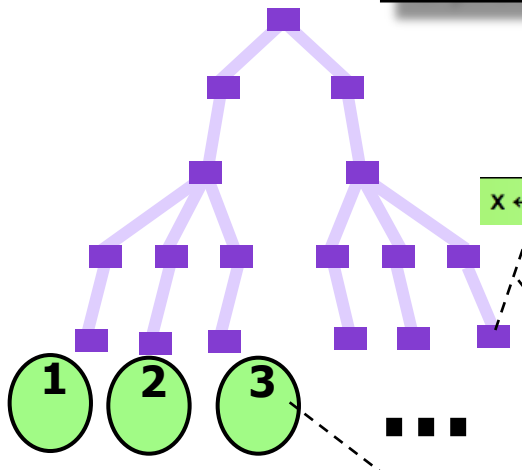
In Indonesian...

Bakar = Spark/Fire
Kiasan = Symbolic

Visualizing Properties of Paths

Verification *(bounded, in some cases)*

Kiasan exhaustively explores all paths through a procedure



paths

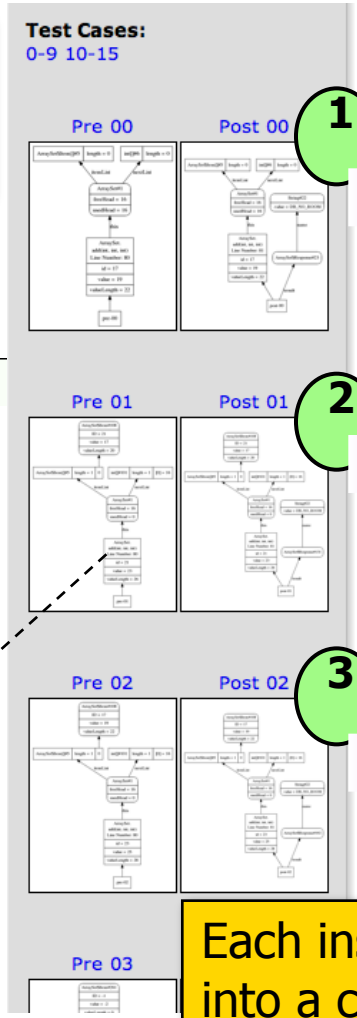
constraints

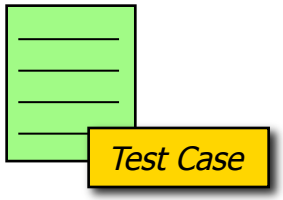
$x \leftarrow \alpha, y \leftarrow \beta, z \leftarrow \pi, \Phi = \{\pi = \alpha + \beta,$

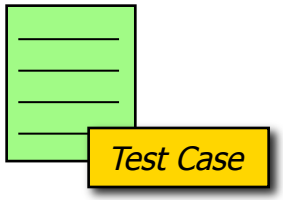
check if post-condition is implied

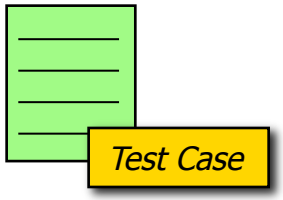
Kiasan builds constraints that characterize each state along each path. These constraints are solved to provide an example ("flow scenarios") of input & output along each path.

Examples / Tests







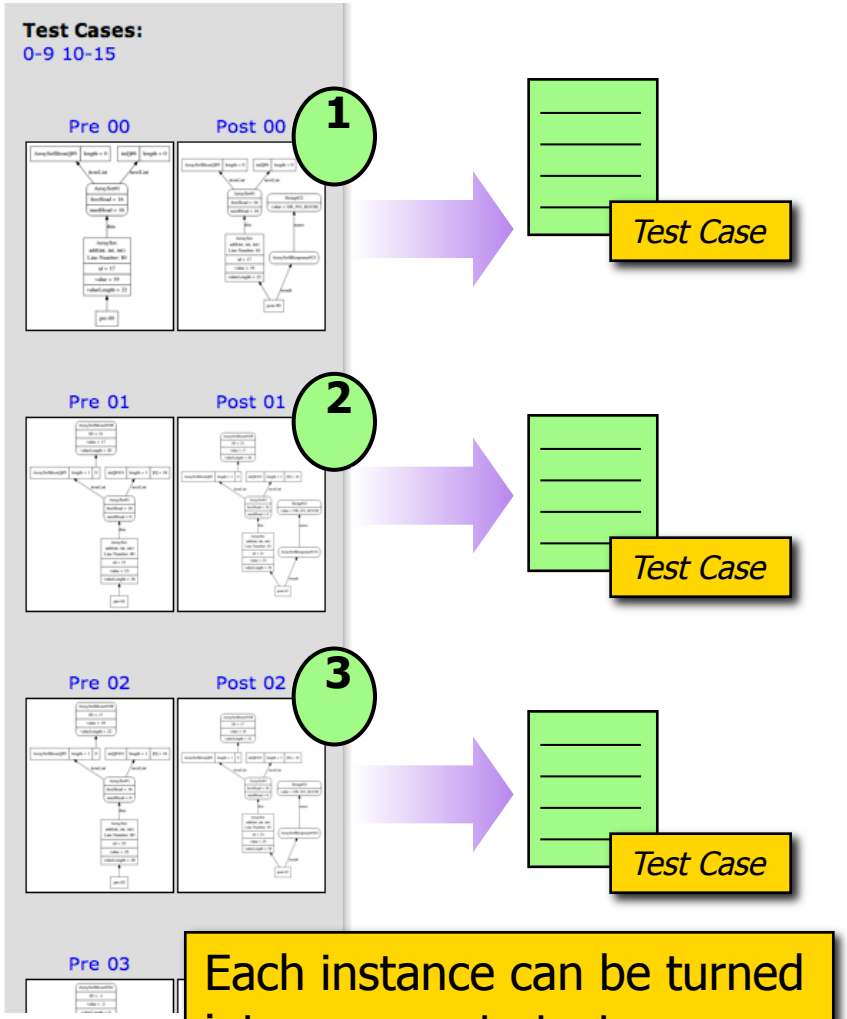


Each instance can be turned into a concrete test case.

Kiasan Output

Why provide examples/tests if we are verifying?

- Tests provide "evidence" to people not familiar with formal methods that something "interesting" is happening in the tool
- When a bug is found along one path, the test provides a counter-example illustrating the bug
- Kiasan's exhaustive exploration automatically yields test suites with very high levels of MCDC coverage



Why Should We Trust?

"I'm worried… 'The sky is purple implies that the moon is made of cheese.'"

In Kiasan, this issue manifests itself to the user as no pre/post pairs being produced

"'Why should we trust your tool?' …social issue. Produce a log of bugs fixed."

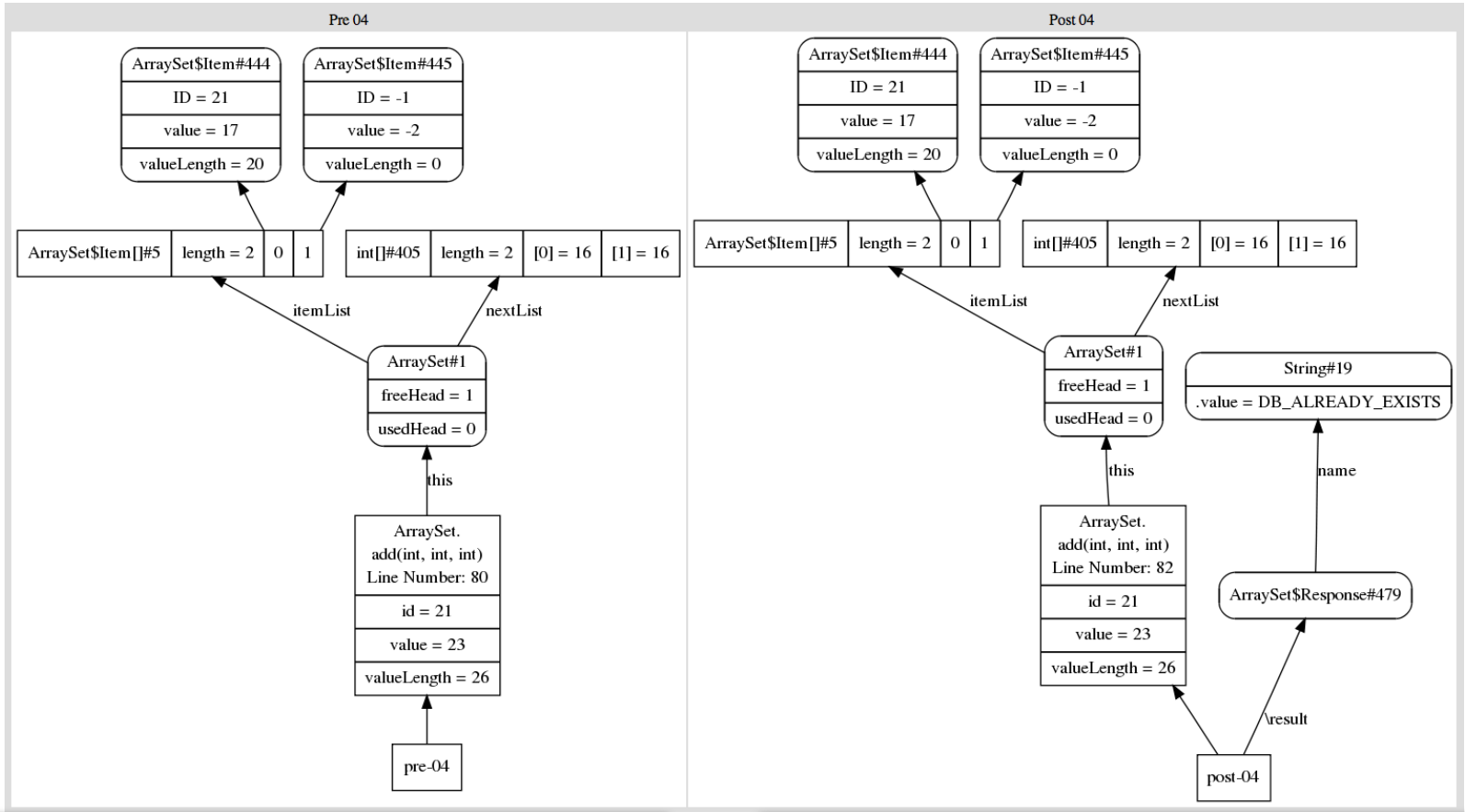
In Kiasan, pre/post-pairs (tests) aim to produce something that developers and auditors can relate to for each verification (our bug list would be rather voluminous ;-)).

"A social issue... Formal methods tools/techniques usually aren't very transparent."

Again, In Kiasan, our pre/post-pairs aim to increase transparency by explaining results in terms of what developers/auditors understand.

Kiasan Output

Sample path input/output for a more complicated example with nested arrays/records



Input -- program state at start of path

Output -- program state at end of path

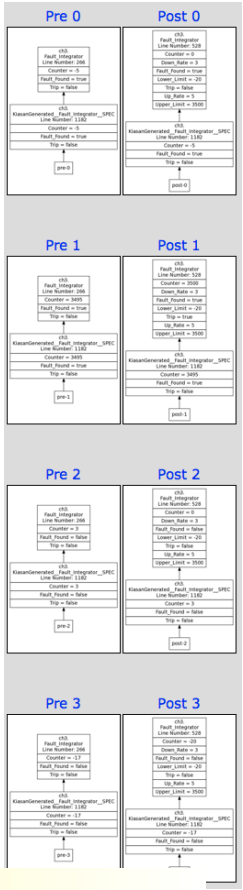
Early Payback

Kiasan does not need contracts to provide useful semantic information – immediately can explore to look for possible run-time exceptions

```
23  procedure Fault_Integrator(Fault_Found : in Boolean;
24                             Trip : in out Boolean;
25                             Counter : in out Integer)
26  is
```

```
37     if Fault_Found then
38       Counter :=
39       if Counter >= Upper_Limit then
40         Trip := True; Counter := Upper_Limit;
41       end if;
42     else
43       Counter := Counter - Down_Rate;
44       if Counter <= Lower_Limit then
45         Trip := False; Counter := Lower_Limit;
46       end if;
47     end if;
48  end Fault_Integrator;
```

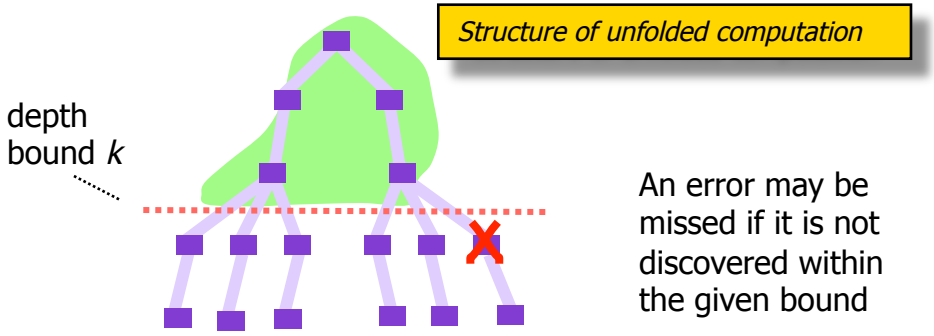
Fully Covered Line



This procedure has four paths, so Kiasan provides four “examples”.

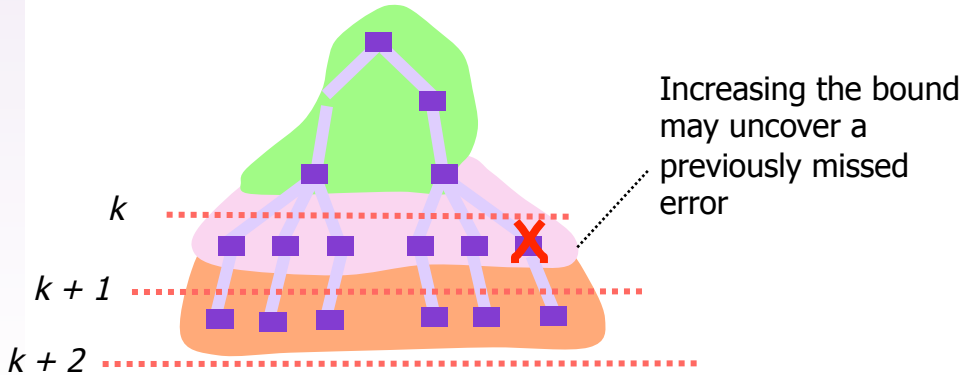
Controlling Cost/Coverage

To ensure the path exploration always terminates, Kiasan uses several bounding techniques which are configurable by the user



- Start with small bounds
- Coverage information provided by the tool indicates if your missing any statements / branches

Increasing *k* increases coverage & cost



- Increase bounds to increase coverage
  - increasing bounds increases time required for analysis
  - run analysis with high bounds for high-confidence as part of over-night regression testing
- Most bugs are found with relatively low bounds

Coverage Information

Package Name: *ArraySet*

Report Rendered: Mon Apr 18 12:36:56 PDT 2011, by Sireum/Kiasan for SPARK v0.1.20100729

Branches Covered For Tests: 23/24 (95.83%) **Branches Covered For Package:** 23/69 (33.33%)

Method Covered:

Percent Ratio

Method	T	E	Instruction Coverage	Branch Coverage	Time
Add	34	0	94.12%	83.33%	0.016s
Delete	10	0	100%	100%	0.050s
Get_Value	10	0	100%	100%	0.020s

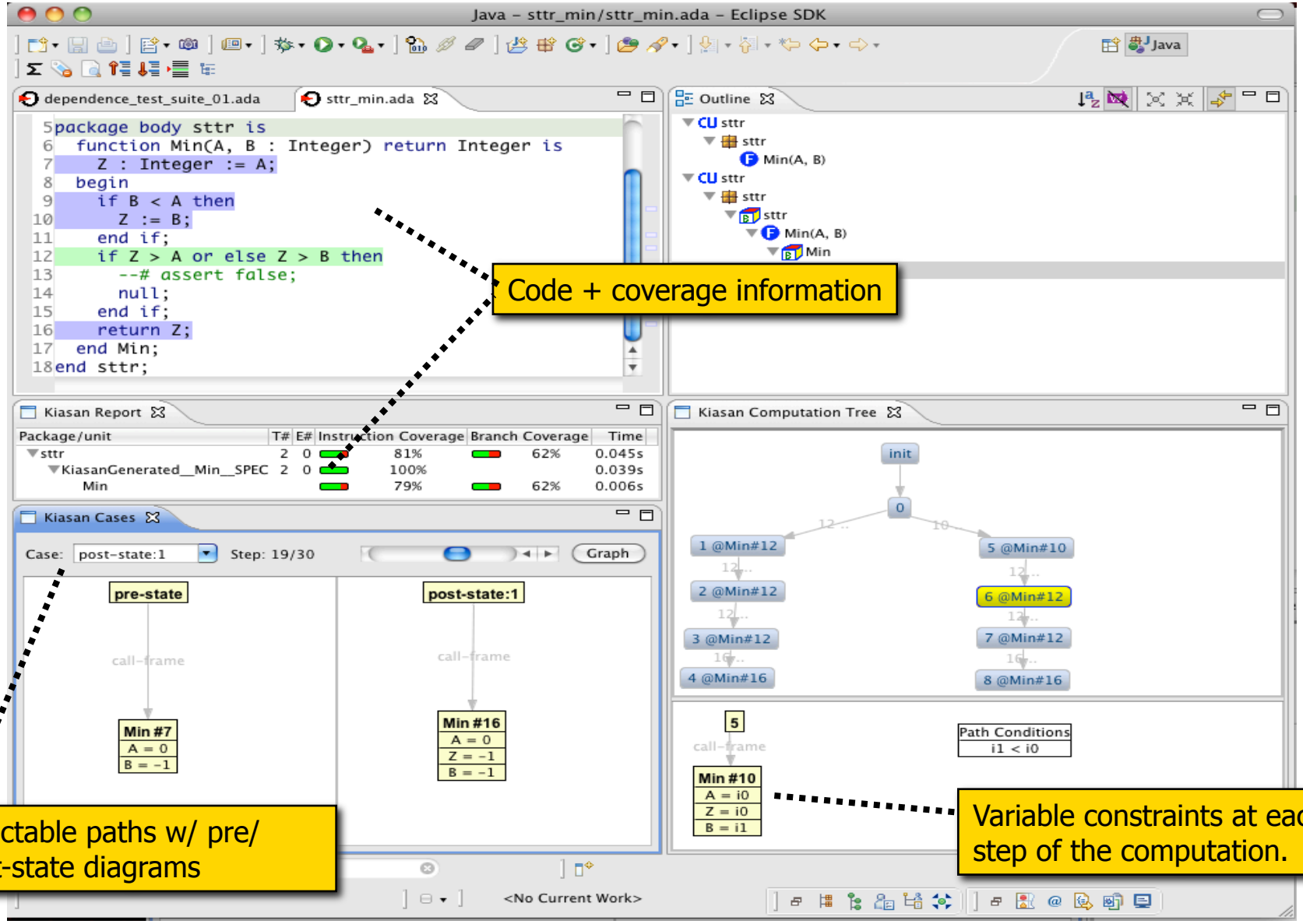
Summary of coverage information

Source Code:

```
1  with ArraySetDefs;
2  with ArraySetUnsigned;
3  use type ArraySetDefs.ID_Type;
4
5  package body ArraySet
6  --# own State is Item_List, Next_List, Free_Head, Used_Head;
7  is
8   Max_Items : constant := 3; -- belt: originally 16
9
10  type Item_Type is record
11    ID          : ArraySetDefs.ID_Type;
12    Value       : ArraySetDefs.Value_Type;
13  end record;
14
15  subtype Item_List_Index_Type is ArraySetUnsigned.Word range 0 .. Max_Items - 1;
16  subtype Link_Type is ArraySetUnsigned.Word range 0 .. Max_Items;
17
18  type Item_List_Type is array (Item_List_Index_Type) of Item_Type;
19  type Next_List_Type is array (Item_List_Index_Type) of Link_Type;
20
21  Item_List : Item_List_Type;
22  Next_List : Next_List_Type;
23
24  Terminator : constant := Link_Type'Last;
25  Inf_Length : constant := Max_Items + 1;
26  Free_Head : Link_Type;
27  Used_Head : Link_Type;
28
29  ------------------Invariant
30
31  ---
32  --- @param head the index of the start of a list (expected to be either
33  ---             Free_Head or Used_Head)
34  --- @return the number of elements reachable from head, or Inf_Length
35  ---         if the list is cyclic.
36  ---
37  function Size_Of_List(head : Link_Type) return ArraySetUnsigned.Word
38    --# global in Next_List;
39  is
40    Cursor : Link_Type;
41    Result : ArraySetUnsigned.Word := 0;
42  begin
43    Cursor := head;
44    while Cursor /= Terminator and Result < Inf_Length loop
45      Result := Result + 1;
```

Source code. Green code indicates executable code that is covered by analysis. Yellow code indicates that code is partially covered (e.g., only one branch of a conditional)

Working at Source Code Level



Benefits of Bakar Kiasan

Bakar Kiasan provides a number of capabilities that help integrate SPARK contract checking directly into developer workflows

- Helpful visualization of paths explored
- Connection to existing quality assurance techniques that developers are familiar with (i.e., testing)
- Capabilities are brought together in the Eclipse IDE
- Technique can be applied very early in the development (even before contracts are written)

Example - Rockwell Collins

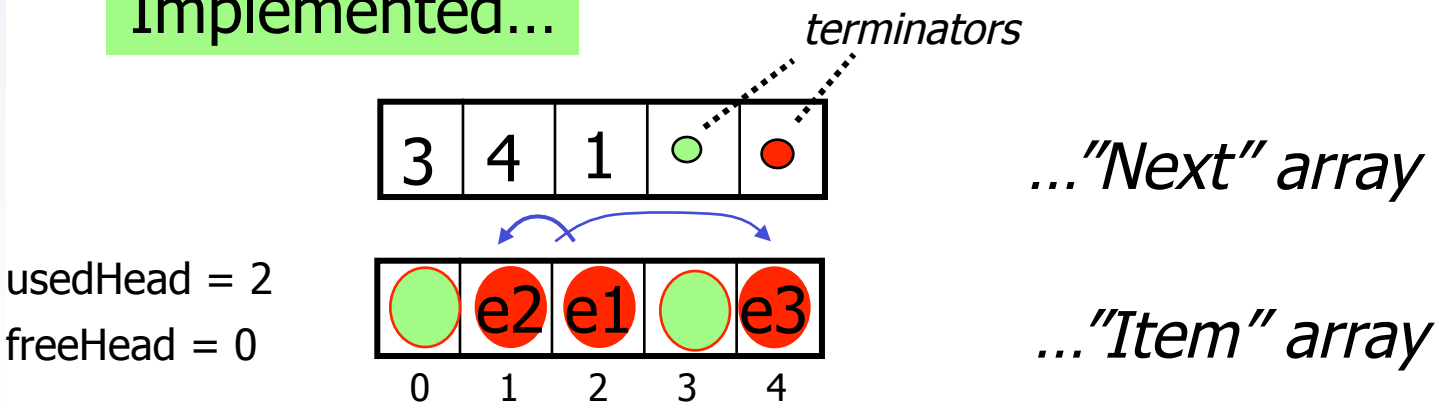
RC ATC engineers have chosen to implement many of the data structures necessary for embedded security devices using array based linked lists.

Conceptually…

used = { e1 e2 e3 } …*database entries*

free = { f1 f2 } …*(behind the scenes) free slots*

Implemented…



…*"Next" array*

…*"Item" array*

Example Properties (excerpts)

Example Representation Invariants

- [Item list] -- IDs are unique and all entries have non-null IDs and values
- [Free list] -- all entries have null IDs and value
- [Item,Free list] -- from each list head (free, item) a terminator is in the set Reachable(*head*)
- [Item, Free list] -- no cycles exist in the item or free lists
- [Item/Free list] -- item and free lists are disjoint and cover all positions in the array

Enhancing Contract Functionality

Existing SPARK contract notation is limited to first-order logic. Plus, any "helper functions" are treated as uninterpreted functions.

```
--# (Response = ... -> (
--#   (for all I in Item_List_Index_Type =>
--#     (ID /= Item_List~(I).ID)) and then
--#   (for some I in Item_List_Index_Type =>
--#     (ID = Item_List(I).ID))))
--#  and then …
```

Example invariants are extremely cumbersome to specify under these limitations

Bakar Kiasan adds support for functions in contracts whose semantics is specified directly using SPARK functions (guaranteed "pure")

```
--# post Invariant(...)
--#   and then
--#   (Response = ... <->
--#     contains(ID, ...) = False)
--#   and then
--#
```

```
function Invariant(...)
  is begin … end
```

```
function contains(...)
  is begin … end
```

…this is also one of the themes of AdaCore Hi-Lite

Example Walkthrough

Delete for Linked Integer Set

```
procedure Delete
  (ID          : in  LinkedIntegerSetDefs.ID_Type;
   Response    : out LinkedIntegerSetDefs.Response_Type)
  --# global in out Item_List, Next_List, Free_Head, Used_Head;
  --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List);
  --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
  --#    and then
  --#    (Response = LinkedIntegerSetDefs.DB_Does_Not_Exist <->
  --#      contains(ID, Used_Head~, Item_List~, Next_List~) = False)
  --#    and then
  --#    (Response = LinkedIntegerSetDefs.DB_Success <->
  --#      (contains(ID, Used_Head~, Item_List~, Next_List~) = True
  --#       and then
  --#       contains(ID, Used_Head, Item_List, Next_List) = False));
```

Example Walkthrough

Delete for Linked Integer Set

```
procedure Delete
   (ID          : in  LinkedIntegerSetDefs.ID_Type;
    Response    : out LinkedIntegerSetDefs.Response_Type)
   --# global in out Item_List, Next_List, Free_Head, Used_Head;
   --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List);
   --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Does_Not_Exist <->
   --#      contains(ID, Used_Head~, Item_List~, Next_List~) = False)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Succes
   --#      (contains(ID, Used_Head~, Item_List~, Ne
   --#       and then
   --#       contains(ID, Used_Head, Item_List, Next_List) = False));
```

Pre-condition: The 'package invariant' is satisfied

Example Walkthrough

Executable code can be accessed from within contracts

```
function Invariant return Boolean
  --# global Next_List, Free_Head, Used_Head, Item_List;
is
begin
  return
    Used_Elements_Invariant and then
    Free_Elements_Invariant and then
    not Is_Cyclic(Free_Head) and then
    not Is_Cyclic(Used_Head) and then
    Size_Of_List(Free_Head) + Size_Of_List(Used_Head) =
      Max_Items;
  end Invariant;
```

Example Walkthrough

Executable code can be accessed from within contracts

```
function Invariant return Boolean
  --# global Next_List, Free_Head, Used_Head, Item_List;
is
begin
  return
    Used_Elements_Invariant and then
    Free_Elements_Invariant and then
    not Is_Cyclic(Free_Head) and then
    not Is_Cyclic(Used_Head) and then
    Size_Of_List(Free_Head) + Size_Of_List(Used_Head) =
      Max_Items;
  end Invariant;
```

Example Walkthrough

Executable code can be accessed from within contracts

```
function Used_Elements_Invariant return Boolean
  --# global Item_List, Next_List, Used_Head;
is
  Cursor : Link_Type;
  Result : Boolean := True;
begin
  Result := Is_Set;
  Cursor := Used_Head;
  while Result and then Cursor /= Terminator loop
    Result := Item_List(Cursor).ID /= LinkedIntegerSetDefs.Null_ID
      and then
      Item_List(Cursor).Value /= LinkedIntegerSetDefs.Null_Value;
    Cursor := Next_List(Cursor);
  end loop;
  return Result;
end Used_Elements_Invariant;
```

Example Walkthrough

Executable code can be accessed from within contracts

```
function Used_Elements_Invariant return Boolean
  --# global Item_List, Next_List, Used_Head;
is
  Cursor : Link_Type;
  Result : Boolean := True;
begin
  Result := Is_Set;
  Cursor := Used_Head;
  while Result and then Cursor /= Terminator loop
    Result := Item_List(Cursor).ID /= LinkedIntegerSetDefs.Null_ID
      and then
      Item_List(Cursor).Value /= LinkedIntegerSetDefs.Null_Value;
    Cursor := Next_List(Cursor);
  end loop;
  return Result;
end Used_Elements_Invariant;
```

Elements reachable from Used_Head must form a set

Example Walkthrough

Executable code can be accessed from within contracts

```
function Used_Elements_Invariant return Boolean
  --# global Item_List, Next_List, Used_Head;
is
  Cursor : Link_Type;
  Result : Boolean := True;
begin
  Result := Is_Set;
  Cursor := Used_Head;
  while Result and then Cursor /= Terminator loop
    Result := Item_List(Cursor).ID /= LinkedIntegerSetDefs.Null_ID
      and then
      Item_List(Cursor).Value /= LinkedIntegerSetDefs.Null_Value;
    Cursor := Next_List(Cursor);
  end loop;
  return Result;
end Used_Elements_Invariant;
```

Elements reachable from Used_Head must have non-null IDs and values

Example Walkthrough

Delete for Linked Integer Set

```
procedure Delete
  (ID          : in  LinkedIntegerSetDefs.ID_Type;
   Response    : out LinkedIntegerSetDefs.Response_Type)
  --# global in out Item_List, Next_List, Free_Head, Used_Head;
  --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List);
  --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
  --#    and then
  --#    (Response = LinkedIntegerSetDefs.DB_Does_Not_Exist <->
  --#      contains(ID, Used_Head~, Item_List~, Next_List~) = False)
  --#    and then
  --#    (Response = LinkedIntegerSetDefs.DB_Success <->
  --#      (contains(ID, Used_Head~, Item_List~, Next_List~) = True
  --#       and then
  --#       contains(ID, Used_Head, Item_List, Next_List) = False));
```

Example Walkthrough

Delete for Linked Integer Set

```
procedure Delete
   (ID          : in  LinkedIntegerSetDefs.ID_Type;
    Response    : out LinkedIntegerSetDefs.Response_Type)
   --# global in out Item_List, Next_List, Free_Head, Used_Head;
   --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List);
   --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Does_Not_Exist <->
   --#      contains(ID, Used_Head~, Item_List~, Next_List~) = False)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Success <->
   --#      (contains(ID, Used_Head~, Item_List~, Next_List~) = True
   --#       and then
   --#       contains(ID, Used_Head, Item_List, Next_List) = False));
```

Case where ID was not in the DB

Example Walkthrough

Delete for Linked Integer Set

```
procedure Delete
   (ID          : in  LinkedIntegerSetDefs.ID_Type;
    Response    : out LinkedIntegerSetDefs.Response_Type)
   --# global in out Item_List, Next_List, Free_Head, Used_Head;
   --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List);
   --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Does_Not_Exist <->
   --#      contains(ID, Used_Head~, Item_List~, Next_List~) = False)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Success <->
   --#      (contains(ID, Used_Head~, Item_List~, Next_List~) = True
   --#       and then
   --#       contains(ID, Used_Head, Item_List, Next_List) = False));
```

Case where ID was found and removed

Linked Integer Set : Add

Add for Linked Integer Set

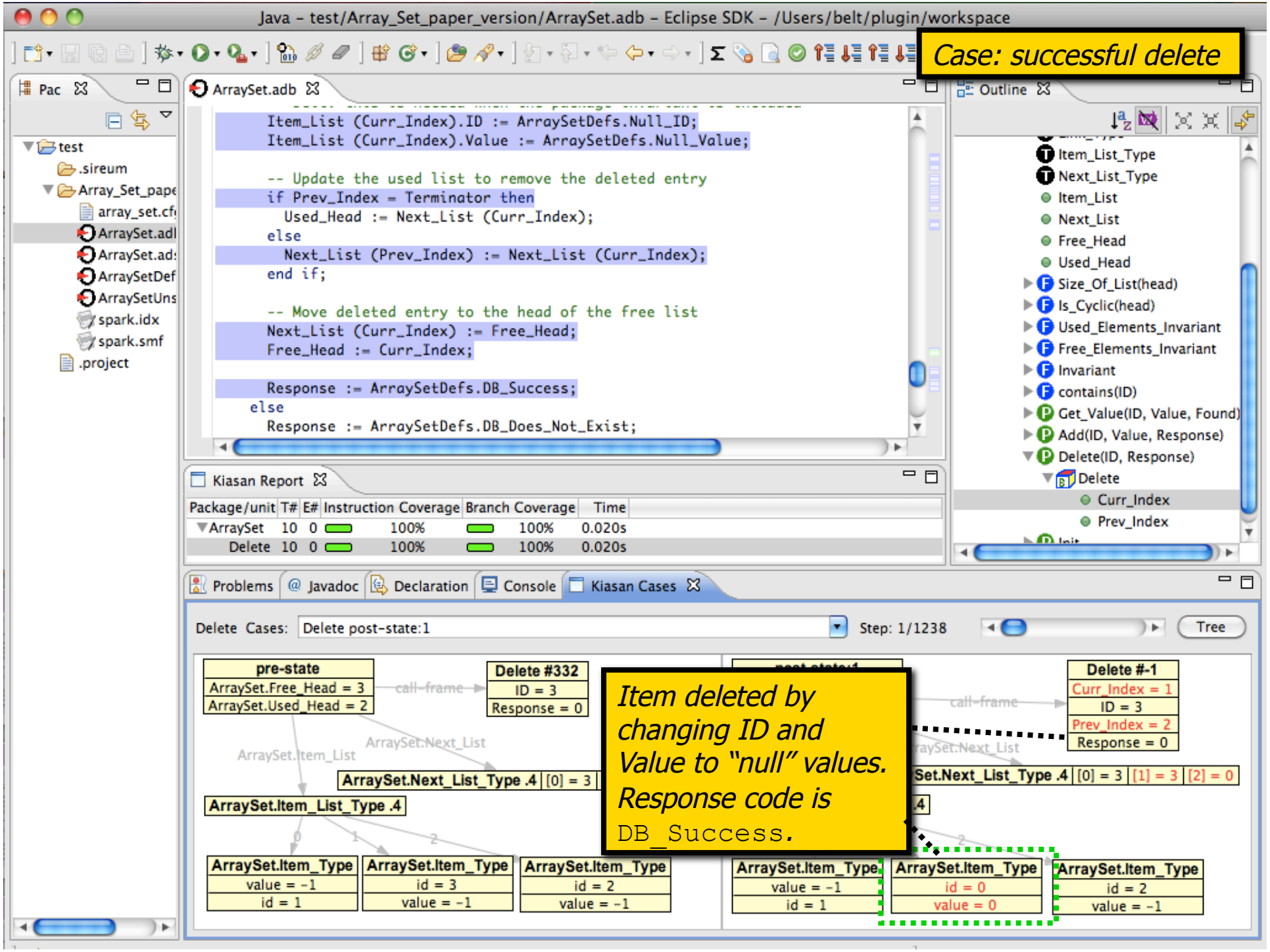
```
procedure Add
   (ID             : in  LinkedIntegerSetDefs.ID_Type;
    Value          : in  LinkedIntegerSetDefs.Value_Type;
    Response       : out LinkedIntegerSetDefs.Response_Type)
   --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#   and then (ID /= LinkedIntegerSetDefs.Null_ID)
   --#   and then (Value /= LinkedIntegerSetDefs.Null_Value);
   --#
   --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#   and then
   --#   (Response = LinkedIntegerSetDefs.DB_Success -> (
   --#     (for all I in Item_List_Index_Type =>
   --#       (ID /= Item_List~(I).ID)) and then
   --#     (for some I in Item_List_Index_Type =>
   --#       (ID = Item_List(I).ID))))
   --#   and then
   --#   (Response = LinkedIntegerSetDefs.DB_Already_Exists -> (
   --#     for some I in Item_List_Index_Type => (
   --#       ID = Item_List~(I).ID and then ID = Item_List(I).ID)))
   --#   and then ...;
```

Linked Integer Set : Add

Add for Linked Integer Set

```
procedure Add
   (ID             : in  LinkedIntegerSetDefs.ID_Type;
    Value          : in  LinkedIntegerSetDefs.Value_Type;
    Response       : out LinkedIntegerSetDefs.Response_Type)
   --# pre Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#    and then (ID /= LinkedIntegerSetDefs.Null_ID)
   --#    and then (Value /= LinkedIntegerSetDefs.Null_Value);
   --#
   --# post Invariant(Next_List, Free_Head, Used_Head, Item_List)
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Success -> (
   --#      (for all I in Item_List_Index_Type =>
   --#        (ID /= Item_List~(I).ID)) and then
   --#      (for some I in Item_List_Index_Type =>
   --#        (ID = Item_List(I).ID))))
   --#    and then
   --#    (Response = LinkedIntegerSetDefs.DB_Already_Exists -> (
   --#      for some I in Item_List_Index_Type => (
   --#        ID = Item_List~(I).ID and then ID = Item_List(I).ID)))
   --#    and then ...;
```

Demo


Java – test/Array_Set_paper_version/ArraySet.adb – Eclipse SDK – /Users/belt/plugin/workspace
Case: successful delete
Item_List (Curr_Index).ID := ArraySetDefs.Null_ID;
Item_List (Curr_Index).Value := ArraySetDefs.Null_Value;
-- Update the used list to remove the deleted entry
if Prev_Index = Terminator then
Used_Head := Next_List (Curr_Index);
else
Next_List (Prev_Index) := Next_List (Curr_Index);
end if;
-- Move deleted entry to the head of the free list
Next_List (Curr_Index) := Free_Head;
Free_Head := Curr_Index;
Response := ArraySetDefs.DB_Success;
else
Response := ArraySetDefs.DB_Does_Not_Exist;
Item_List_Type
Next_List_Type
Item_List
Next_List
Free_Head
Used_Head
Size_Of_List(head)
Is_Cyclic(head)
Used_Elements_Invariant
Free_Elements_Invariant
Invariant
contains(ID)
Get_Value(ID, Value, Found)
Add(ID, Value, Response)
Delete(ID, Response)
Delete
Curr_Index
Prev_Index
Package/unit T# E# Instruction Coverage Branch Coverage Time
ArraySet 10 0 100% 100% 0.020s
Delete 10 0 100% 100% 0.020s
Problems Javadoc Declaration Console Kiasan Cases
Delete Cases: Delete post-state:1
Step: 1/1238
Tree
pre-state
ArraySet.Free_Head = 3
ArraySet.Used_Head = 2
call-frame
Delete #332
ID = 3
Response = 0
ArraySet.Item_List
ArraySet.Next_List
ArraySet.Next_List_Type .4 [0] = 3
ArraySet.Item_List_Type .4
ArraySet.Item_Type
value = -1
id = 1
ArraySet.Item_Type
id = 3
value = -1
ArraySet.Item_Type
id = 2
value = -1
Item deleted by changing ID and Value to "null" values. Response code is DB_Success.
Delete #-1
Curr_Index = 1
ID = 3
Prev_Index = 2
Response = 0
Set.Next_List_Type .4 [0] = 3 [1] = 3 [2] = 0
ArraySet.Item_Type
value = -1
id = 1
ArraySet.Item_Type
id = 0
value = 0
ArraySet.Item_Type
id = 2
value = -1

ArraySet.Delete 1

```
function Delete_1_embedded return Boolean is
  Response_OUT : Response_Type;
begin
  ArraySet.Free_Head := 3;
  ArraySet.Used_Head := 2;
  ArraySet.Item_List :=
    ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                             1 => ArraySet.Item_Type'(id => 3, value => -1),
                             2 => ArraySet.Item_Type'(id => 2, value => -1));
  ArraySet.Next_List := ArraySet.Next_List_Type'(0 => 3, 1 => 0, 2 => 1);

  ArraySet.Delete (ID => 3, Response => Response_OUT);

  return (ArraySet.Free_Head = 1) and then (ArraySet.Used_Head = 2)
    and then (Response_OUT = 0)
    and then (ArraySet.Item_List =
      ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                               1 => ArraySet.Item_Type'(id => 0, value => 0),
                               2 => ArraySet.Item_Type'(id => 2, value => -1))
    and then (ArraySet.Next_List = ArraySet.Next_List_Type'(0 => 3, 1 => 3, 2 => 0));
end Delete_1_embedded;
```

ArraySet.Delete 1

Initialize Pre-State

```
function Delete_1_embedded return Boolean is
  Response_OUT : Response_Type;
begin
  ArraySet.Free_Head := 3;
  ArraySet.Used_Head := 2;
  ArraySet.Item_List :=
    ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                             1 => ArraySet.Item_Type'(id => 3, value => -1),
                             2 => ArraySet.Item_Type'(id => 2, value => -1));
  ArraySet.Next_List := ArraySet.Next_List_Type'(0 => 3, 1 => 0, 2 => 1);

  ArraySet.Delete (ID => 3, Response => Response_OUT);

  return (ArraySet.Free_Head = 1) and then (ArraySet.Used_Head = 2)
    and then (Response_OUT = 0)
    and then (ArraySet.Item_List =
      ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                               1 => ArraySet.Item_Type'(id => 0, value => 0),
                               2 => ArraySet.Item_Type'(id => 2, value => -1))
    and then (ArraySet.Next_List = ArraySet.Next_List_Type'(0 => 3, 1 => 3, 2 => 0));
end Delete_1_embedded;
```

ArraySet.Delete 1

```
function Delete_1_embedded return Boolean is
  Response_OUT : Response_Type;
begin
  ArraySet.Free_Head := 3;
  ArraySet.Used_Head := 2;
  ArraySet.Item_List :=
    ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                             1 => ArraySet.Item_Type'(id => 3, value => -1),
                             2 => ArraySet.Item_Type'(id => 2, value => -1));
  ArraySet.Next_List := ArraySet.Next_List_Type'(0 => 3, 1 => 0, 2 => 2);

  ArraySet.Delete (ID => 3, Response => Response_OUT);

  return (ArraySet.Free_Head = 1) and then (ArraySet.Used_Head = 2)
    and then (Response_OUT = 0)
    and then (ArraySet.Item_List =
      ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                               1 => ArraySet.Item_Type'(id => 0, value => 0),
                               2 => ArraySet.Item_Type'(id => 2, value => -1))
    and then (ArraySet.Next_List = ArraySet.Next_List_Type'(0 => 3, 1 => 3, 2 => 0));
end Delete_1_embedded;
```

Invoke Delete

ArraySet.Delete 1

```
function Delete_1_embedded return Boolean is
  Response_OUT : Response_Type;
begin
  ArraySet.Free_Head := 3;
  ArraySet.Used_Head := 2;
  ArraySet.Item_List :=
    ArraySet.Item_List_Typ                    ype'(id => 1, value => -1),
                                              ype'(id => 3, value => -1),
                                              ype'(id => 2, value => -1));
  ArraySet.Next_List := Ar                    => 3, 1 =>

  ArraySet.Delete (ID => 3                    _OUT);

  return (ArraySet.Free_Head = 1) and then (ArraySet.Used_Head = 2)
    and then (Response_OUT = 0)
    and then (ArraySet.Item_List =
      ArraySet.Item_List_Type'(0 => ArraySet.Item_Type'(id => 1, value => -1),
                               1 => ArraySet.Item_Type'(id => 0, value => 0),
                               2 => ArraySet.Item_Type'(id => 2, value => -1))
    and then (ArraySet.Next_List = ArraySet.Next_List_Type'(0 => 3, 1 => 3, 2 => 0));
end Delete_1_embedded;
```

Item deleted by changing ID and Value to "null" values. Response code is `DB_Success`.

Check Post-Condition

Performance Evaluation

- Can we provide a significant increase in automation of contract checking over existing SPARK tools?
- Scalability
  - Can be applied during the typical code/test/debug loop
  - Can the technique scale to sizes of data structures that would be reasonable for embedded systems

Experimental Results

Scalability *(k = array size)*

Time (secs)

- Easily fits in the code/test/debug loop for small k
- Previous Rockwell Collins approach using Prover maxed out at k=3
  - Plus, Kiasan works directly on SPARK code and doesn't require manual translation

Package.Procedure Name	VC	k=3	k=4	k=5	k=6	k=7	k=8
Sort.Bubble	13/18	0.17	0.96	2.09	8.43	71.72	890.18
Sort.Insertion	10/14	0.15	0.98	2.06	8.24	70.72	892.17
Sort.Selection	28/30	0.16	1.06	2.28	9.95	90.14	1356.18
Sort.Shell	17/18	0.15	0.98	2.12	8.47	74.09	941.99
IntegerSet.Get_Element_Index	8/11	0.04	0.05	0.06	0.07	0.08	0.10
IntegerSet.Add	3/5	0.24	0.44	0.62	0.79	0.80	1.04
IntegerSet.Remove	5/6	0.16	0.30	0.56	0.96	1.21	1.36
IntegerSet.Empty	3/3	0.02	0.02	0.02	0.02	0.02	0.02
LinkedIntegerSet.Get_Value	9/10	0.64	0.88	1.13	1.51	2.19	2.85
LinkedIntegerSet.Add	14/16	0.43	0.73	1.66	5.26	34.96	379.34
LinkedIntegerSet.Delete	18/21	0.52	0.72	1.03	1.56	2.10	2.75
LinkedIntegerSet.Init	16/17	0.05	0.04	0.04	0.05	0.05	0.05
MMR.Fill_Mem_Row	8/10	0.18					
MMR.Zero_Mem_Row	6/7	0.19					
MMR.Zero_Flags	6/7	0.05					
MMR.Read_Msgs	3/4	1.71					
MMR.Send_Msg	4/5	0.50					
MMR.Route	62/67	13.90					

Experimental Results

Scalability *(k = array size)*

Time (secs)

- Scales well for methods which don't contain quantification
- Larger bounds can be explored as part of a nightly regression test process
- Implementation can be easily distributed

Package.Procedure Name	VC	k=3	k=4	k=5	k=6	k=7	k=8
Sort.Bubble	13/18	0.17	0.96	2.09	8.43	71.72	890.18
Sort.Insertion	10/14	0.15	0.98	2.06	8.24	70.72	892.17
Sort.Selection	28/30	0.16	1.06	2.28	9.95	90.14	1356.18
Sort.Shell	17/18	0.15	0.98	2.12	8.47	74.09	941.99
IntegerSet.Get_Element_Index	8/11	0.04	0.05	0.06	0.07	0.08	0.10
IntegerSet.Add	3/5	0.24	0.44	0.62	0.79	0.80	1.04
IntegerSet.Remove	5/6	0.16	0.30	0.56	0.96	1.21	1.36
IntegerSet.Empty	3/3	0.02	0.02	0.02	0.02	0.02	0.02
LinkedIntegerSet.Get_Value	9/10	0.64	0.88	1.13	1.51	2.19	2.85
LinkedIntegerSet.Add	14/16	0.43	0.73	1.66	5.26	34.96	379.34
LinkedIntegerSet.Delete	18/21	0.52	0.72	1.03	1.56	2.10	2.75
LinkedIntegerSet.Init	16/17	0.05	0.04	0.04	0.05	0.05	0.05
MMR.Fill_Mem_Row	8/10	0.18					
MMR.Zero_Mem_Row	6/7	0.19					
MMR.Zero_Flags	6/7	0.05					
MMR.Read_Msgs	3/4	1.71					
MMR.Send_Msg	4/5	0.50					
MMR.Route	62/67	13.90					

Kiasan Methodology

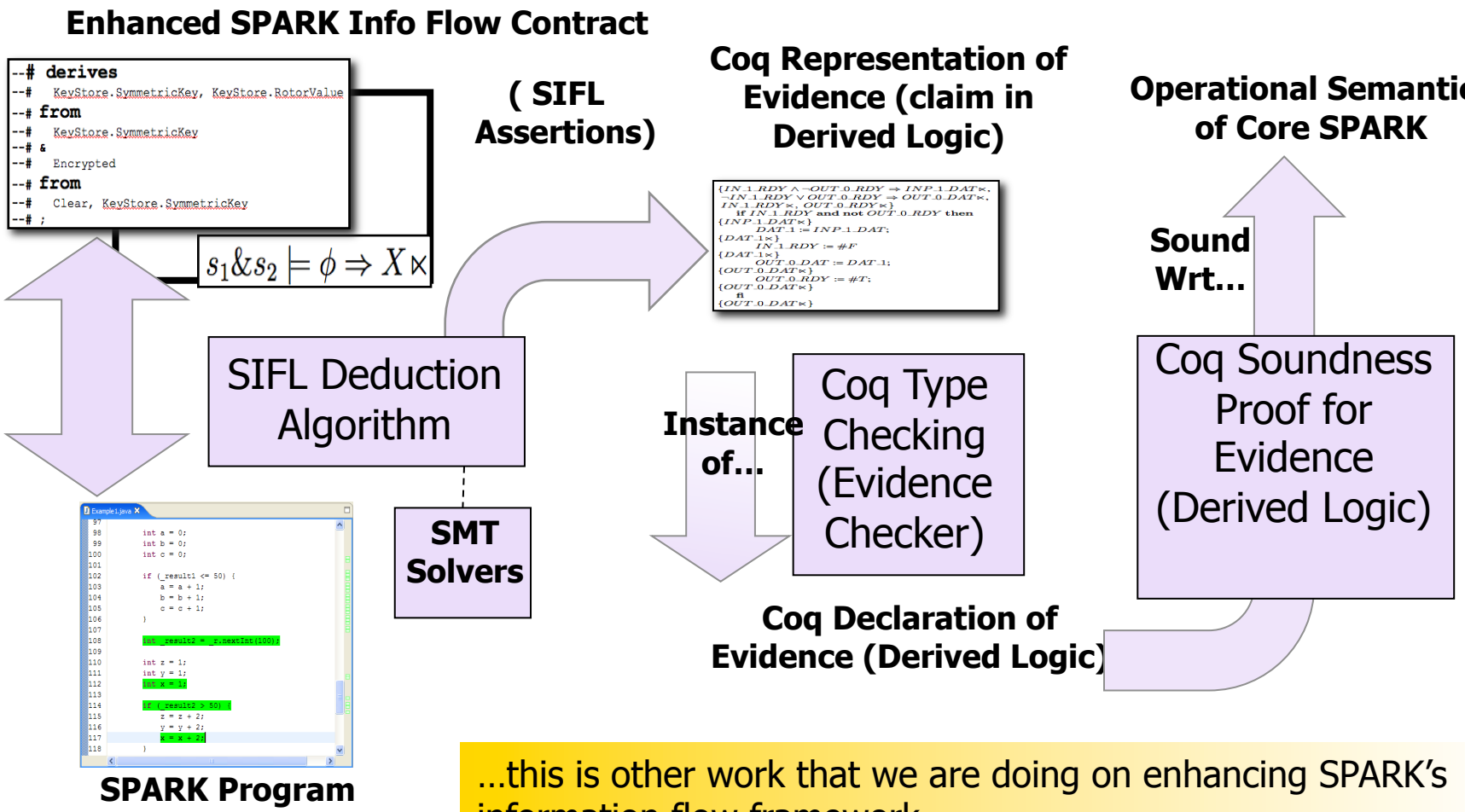
- Checking in IDE
  - start with small bounds
  - incrementally check
  - scenario and test case generation for violations
- More exhaustive checking
  - higher bounds with overnight/parallel checking
  - Kiasan tells you if coverage criteria has been met
- Code understanding
  - select any block of code,
    Kiasan generates flow scenarios giving path coverage
- Test case generation for regression testing
  - automatically generate tests (full MCDC coverage) from code
- Add loop invariants for complete verification

Summary

- Considered the challenge of contract checking in SPARK – one of the best commercially supported frameworks for code-level safety critical software development
- Demonstrated how symbolic execution can potentially change the status quo for the use of SPARK contracts
  - **From** very rarely used
    to useable by typical developers within the normal code-test-debug loop
- Demonstrated how Bakar Kiasan can provide checking of very complex contracts from embedded security applications
  - Flexible combination of declarative and executable specs
- Illustrated multiple ways of leveraging the information produced by symbolic execution
  - Pre/post-state visualization
  - Test case generation
  - Bridging the gap between formal methods and testing

Verification/Evidence Framework

Another idea – Tools can produce independently auditable and machine checked proofs of correctness



...this is other work that we are doing on enhancing SPARK's information flow framework

Next Steps…

SAnToS Laboratory,
Kansas State University
http://www.santoslab.org

- Around 90% of SPARK handled – extend to 100%
- Enhancements to SPARK contract language
  - Package invariants, data abstractions & refinement
- Integration with AdaCore Hi-Lite infrastructure & tool chain
- Evaluation in Rockwell Collins research projects